pysimavr Documentation

Release 0.0.3

ponty

October 24, 2011

CONTENTS

pysimavr

Date October 24, 2011

PDF pysimavr.pdf

Contents:

pysimavr is a python wrapper for simavr which is AVR and arduino simulator

Links:

- home: https://github.com/ponty/pysimavr
- documentation: http://ponty.github.com/pysimavr

Features:

- python wrapper using swig
- simavr is included for easier installation
- object oriented interface on top of the generated interface
- maximum speed can be real-time
- serial communication
- check simavr documentation

Known problems:

- Python 3 is not supported
- tested only on linux
- more tests needed
- PWM simulation is not real-time
- missing PWM modes
- a lot of messages on stdout
- LCD simulator is not fully implemented

Possible usage:

- unit test
- simulator

Similar projects:

- simavr
- emulino
- Arduino Unit
- arduemu

CHAPTER

ONE

BASIC USAGE

```
>>> from pysimavr.avr import Avr
>>> avr=Avr(mcu='atmega48',f_cpu=8000000)
>>> firmware = Firmware('lcd.elf')
>>> avr.load_firmware(firmware)

>>> from pysimavr.sim import ArduinoSim
>>> print ArduinoSim(snippet='Serial.print("hello!");').get_serial()
hello!
```

CHAPTER

TWO

INSTALLATION

check simavr doc: http://gitorious.org/simavr/pages/GetStarted

ignore these in simavr doc:

- OpenGl (freeglut)
- gcc-avr
- avr-libc
- make

2.1 General

- install python
- install setuptools
- install swig (for source build only)
- install header files and a static library for Python (for source build only)
- install a compiler (for source build only)
- install elf library
- install the program:

```
# as root
easy_install pysimavr
```

2.2 Ubuntu

```
sudo apt-get install python-setuptools
sudo apt-get install swig
sudo apt-get install python-dev
sudo apt-get install gcc
sudo apt-get install libelf-dev
sudo easy_install pysimavr
```

2.3 Uninstall

first install pip:

```
# as root
pip uninstall pysimavr
```

CHAPTER

THREE

USAGE

pysimavr.examples.simple:

```
from pysimavr.avr import Avr

avr=Avr(mcu='atmega48',f_cpu=8000000)
avr.step(1)
print avr.pc
```

```
$ python -m pysimavr.examples.simple
Starting atmega48 - flashend 0fff ramend 02ff e2end 00ff
atmega48 init
2
```

pysimavr.examples.hello:

```
from pysimavr.sim import ArduinoSim
from entrypoint2 import entrypoint

@entrypoint
def run_sim():
    print ArduinoSim(snippet='Serial.print("hello!");').get_serial()
```

```
$ python -m pysimavr.examples.hello
Loaded 2148 .text
Loaded 26 .data
Starting atmega328 - flashend 7fff ramend 08ff e2end 03ff
atmega328 init
uart_udp_init bridge on port 4321
avr_timer_reconfigure-0 clock turned off
avr_timer_reconfigure-0 clock turned off
avr_timer_configure-0 TOP 7812.50Hz = 2048 cycles
avr_timer_configure-0 C 8888.89Hz = 1800 cycles
avr_timer_configure-0 TOP 976.56Hz = 16384 cycles
avr_timer_configure-0 C 1111.11Hz = 14400 cycles
avr_timer_configure-1 TOP 30.52Hz = 524288 cycles
avr_timer_configure-1 C 8888.89Hz = 1800 cycles
avr_timer_configure-1 TOP 3.81Hz = 4194304 cycles
avr_timer_configure-1 C 1111.11Hz = 14400 cycles
avr_timer_configure-1 TOP 976.56Hz = 16384 cycles
avr_timer_configure-1 C 1111.11Hz = 14400 cycles
avr_timer_configure-2 TOP 976.56Hz = 16384 cycles
avr_timer_configure-2 C 1111.11Hz = 14400 cycles
avr_timer_configure-2 TOP 976.56Hz = 16384 cycles
avr_timer_configure-2 C 1111.11Hz = 14400 cycles
```

```
ADC Start AREF 0 AVCC 5000
UART-0 configured to 00cf = 4807 bps, 5 data 1 stop
Roughtly 1666 usec per bytes
hello!
```

3.1 vcd export example

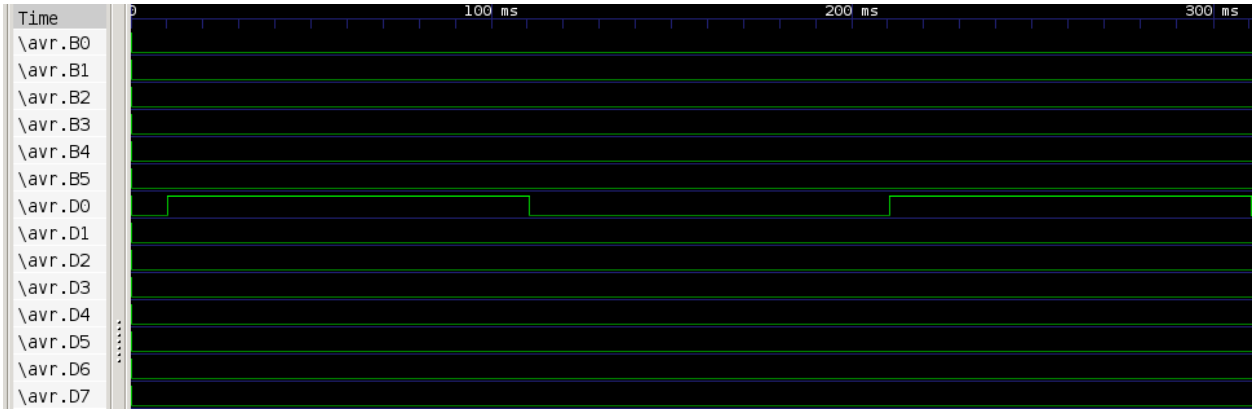
pysimavr.examples.vcd:

```
from entrypoint2 import entrypoint
from pysimavr.sim import ArduinoSim

@entrypoint
def run_sim(vcdfile='delay.vcd'):
    snippet='''
        Serial.println("start");
        pinMode(0, OUTPUT);
        digitalWrite(0, HIGH);
        delay(100);
        digitalWrite(0, LOW);
        delay(100);
        digitalWrite(0, HIGH);
        delay(100);
        digitalWrite(0, LOW);
        delay(100);
        Serial.println("end");
    '''
    sim=ArduinoSim(snippet=snippet, vcd=vcdfile, timespan=0.5)
    sim.run()
    print sim.serial
```

```
>>> from pysimavr.examples.vcd import run_sim
>>> run_sim(vcdfile='docs/vcd.vcd')
Loaded 2954 .text
Loaded 30 .data
Starting atmega328 - flashend 7fff ramend 08ff e2end 03ff
atmega328 init
uart_udp_init bridge on port 4321
avr_timer_reconfigure-0 clock turned off
avr_timer_reconfigure-0 clock turned off
avr_timer_configure-0 TOP 7812.50Hz = 2048 cycles
avr_timer_configure-0 C 15151.52Hz = 1055 cycles
avr_timer_configure-0 TOP 976.56Hz = 16384 cycles
avr_timer_configure-0 C 1893.94Hz = 8447 cycles
avr_timer_configure-1 TOP 30.52Hz = 524288 cycles
avr_timer_configure-1 C 15151.52Hz = 1055 cycles
avr_timer_configure-1 TOP 3.81Hz = 4194304 cycles
avr_timer_configure-1 C 1893.94Hz = 8447 cycles
avr_timer_configure-1 TOP 976.56Hz = 16384 cycles
avr_timer_configure-1 C 1893.94Hz = 8447 cycles
avr_timer_configure-2 TOP 976.56Hz = 16384 cycles
avr_timer_configure-2 C 1893.94Hz = 8447 cycles
avr_timer_configure-2 TOP 976.56Hz = 16384 cycles
avr_timer_configure-2 C 1893.94Hz = 8447 cycles
ADC Start AREF 0 AVCC 5000
UART-0 configured to 00cf = 4807 bps, 5 data 1 stop
```

```
Roughtly 1666 usec per bytes
start
end
```



3.2 unit test example

pysimavr/examples/test_example.py

```
''' unit test example'''

from pysimavr.sim import ArduinoSim

def test_atmega88():
    mcu = 'atmega88'
    snippet = 'Serial.print("hi");'

    output = ArduinoSim(snippet=snippet, mcu=mcu, timespan=0.01).get_serial()
    assert output == 'hi'
```

```
$ nosetests --verbose pysimavr/examples/test_example.py
pysimavr.examples.test_example.test_atmega88 ... ok

----------------------------------------------------------------------
Ran 1 test in 2.216s

OK
Loaded 2068 .text
Loaded 22 .data
Starting atmega88 - flashend 1fff ramend 04ff e2end 01ff
atmega88 init
uart_udp_init bridge on port 4321
avr_timer_reconfigure-0 clock turned off
avr_timer_reconfigure-0 clock turned off
avr_timer_configure-0 TOP 7812.50Hz = 2048 cycles
avr_timer_configure-0 C 10638.30Hz = 1504 cycles
avr_timer_configure-0 TOP 976.56Hz = 16384 cycles
avr_timer_configure-0 C 1329.79Hz = 12032 cycles
avr_timer_configure-1 TOP 30.52Hz = 524288 cycles
avr_timer_configure-1 C 10638.30Hz = 1504 cycles
avr_timer_configure-1 TOP 3.81Hz = 4194304 cycles
avr_timer_configure-1 C 1329.79Hz = 12032 cycles
avr_timer_configure-1 TOP 976.56Hz = 16384 cycles
```

```
avr_timer_configure-1 C 1329.79Hz = 12032 cycles
avr_timer_configure-2 TOP 976.56Hz = 16384 cycles
avr_timer_configure-2 C 1329.79Hz = 12032 cycles
avr_timer_configure-2 TOP 976.56Hz = 16384 cycles
avr_timer_configure-2 C 1329.79Hz = 12032 cycles
ADC Start AREF 0 AVCC 5000
UART-0 configured to 00cf = 4807 bps, 5 data 1 stop
Roughtly 1666 usec per bytes
```

CHAPTER

FOUR

FILE HIERARCHY

```
|-docs                  sphinx documentation
|---_build              generated documentation
|-pysimavr              main python package, high level classes
|---examples            examples
|---swig                all swig files (simavr and parts)
|-----cores             copy from simavr
|-----include           copy from simavr
|-------avr             copy from avr-libc
|-----parts             some electronic parts in c
|-----sim               copy from simavr
|-tests                 unit tests
```

CHAPTER

FIVE

API

There are 2 interfaces:

- pysimavr.swig.*: low level, generated by swig
- pysimavr.*: high level classes, they can redirect function calls to low level interface. Example: Avr class (high level) has all properties and methods of avr_t class (low level) automatically.

5.1 low level interface

class pysimavr.swig.ac_input.**ac_input_t**

avr

irq

value

class pysimavr.swig.hd44780.**hd44780_t**

avr

cursor

datapins

flags

h

irq

pinstate

readpins

vram

w

class pysimavr.swig.inverter.**inverter_t**

avr

irq

out

class pysimavr.swig.ledrow.**ledrow_t**

avr

irq

pinstate

pinstate_changed

class pysimavr.swig.sgm7.**sgm7_t**

avr

digit_count

digit_pin

digit_port

digit_segments

digit_segments_changed

irq

pinstate

segment_pin

segment_port

class pysimavr.swig.simavr.**avr_io_t**

avr

dealloc

ioctl

irq

irq_count

irq_ioctl_get

irq_names

kind

next

reset

class pysimavr.swig.simavr.**avr_ioport_getirq_t**

bit

irq

class pysimavr.swig.simavr.**avr_ioport_state_t**

ddr

name

pin

port

class pysimavr.swig.simavr.**avr_ioport_t**

io

name

pcint

r_ddr

r_pcint

r_pin

r_port

class pysimavr.swig.simavr.**avr_irq_pool_t**

count

irq

class pysimavr.swig.simavr.**avr_irq_t**

flags

hook

irq

name

pool

value

class pysimavr.swig.simavr.**avr_kind_t**

make

names

class pysimavr.swig.simavr.**avr_symbol_t**

addr

symbol

class pysimavr.swig.simavr.**avr_t**

aref

avcc

codeend

cycle

cycle_timer

cycle_timer_map

data

e2end

eind

flash

flashend

frequency

fuse

gdb

gdb_port

i_shadow

init

io

io_port

io_shared_io

io_shared_io_count

irq_pool

mmcu

next_cycle_timer

pc

pending

pending_wait

ramend

rampz

reset

run

signature

sleep

special_deinit

special_init

sreg

state

trace

trace_data

vcc

vcd

vector

vector_size

class pysimavr.swig.simavr.**avr_t_cycle_timer**

param

timer

when

class pysimavr.swig.simavr.**avr_t_io**

irq

r

w

class pysimavr.swig.simavr.**avr_t_io_r**

c

param

class pysimavr.swig.simavr.**avr_t_io_shared_io**

io

used

class pysimavr.swig.simavr.**avr_t_io_shared_io_io**

c

param

class pysimavr.swig.simavr.**avr_t_io_w**

c

param

class pysimavr.swig.simavr.**avr_trace_data_t**

codeline

old

old_pci

touched

class pysimavr.swig.simavr.**avr_trace_data_t_old**

pc

sp

class pysimavr.swig.simavr.**avr_vcd_log_t**

signal

value

when

class pysimavr.swig.simavr.**avr_vcd_signal_t**

alias

irq

name

size

class pysimavr.swig.simavr.**avr_vcd_t**

avr

filename

log

logindex

output

period

signal

signal_count

start

class pysimavr.swig.simavr.**elf_firmware_t**

aref

avcc

bsssize

codeline

codesize

command_register_addr

console_register_addr

datasize

eeprom

eesize

flash

flashbase

flashsize

frequency

mmcu

trace

tracecount

tracename

traceperiod

vcc

class pysimavr.swig.simavr.**elf_firmware_t_trace**

addr

mask

name

5.2 high level interface

class pysimavr.ac.**Ac**(*avr*)

getirq(*pin*)

class pysimavr.avr.**Avr**(*firmware=None*, *mcu=None*, *f_cpu=None*, *avcc=5*, *vcc=5*)

arduino_targets = ['atmega48', 'atmega88', 'atmega168', 'atmega328p']

avcc

fpeek(*addr*)

getirq(*pin*)

goto_cycle(*n*)

goto_time(*tsec*)

load_firmware(*firmware*)

move_time_marker(*tsec_diff*)

pause()

peek(*addr*)

reset()

run()

states = ['Limbo', 'Stopped', 'Running', 'Sleeping', 'StepStepDone']

step(*n=1*, *sync=True*)

terminate()

time_passed()

vcc

exception pysimavr.avr.**UnkwownAvrError**

pysimavr.connect.**connect_irqs**(*irq_out*, *irq_in*, *bidirectional=False*)

pysimavr.connect.**connect_pins_by_rule**(*rule*, *device_map*, *vcd=None*)

rule example:

B0 –> D4 -> vcd

B1 <== D5 B2 => D6 #B3 <=> D7

class pysimavr.firmware.**Firmware**(*filename=None*)

mcu

read(*filename*)

class pysimavr.inverter.**Inverter**(*avr*)

getirq(*pin*)

out(*i*)

class pysimavr.lcd.**Lcd**(*avr*, *size=(20, 2)*)

get_char(*x*, *y*)

getirq(*pin*)

pinstate(*pin*)

reset()

class pysimavr.ledrow.**LedRow**(*avr*, *size=8*)

getirq(*pin*)

pinstate(*i*)

reset_dirty(*i*)

read and reset

class pysimavr.sgm7.**Sgm7**(*avr*, *size=4*)

digit_segments(*digit_index*)

getirq(*pin*)

pinindex(*pin_name*)

pinstate(*pin*)

reset_dirty(*digit_index*)

read and reset

class pysimavr.vcdfile.**VcdFile**(*avr*, *filename='gtkwave_output.vcd'*, *period=10*)

add_signal(*irq*, *name=None*, *bits=1*)

start()

stop()

terminate()

CHAPTER

SIX

DEVELOPMENT

6.1 Tools

1. setuptools
2. Paver
3. nose
4. ghp-import
5. pyflakes
6. pychecker
7. paved fork
8. Sphinx
9. sphinxcontrib-programscreenshot
10. sphinxcontrib-paverutils
11. `autorun` from sphinx-contrib (there is no simple method, you have to download/unpack/setup)

6.2 Install on ubuntu

```
sudo apt-get install python-setuptools
sudo apt-get install python-paver
sudo apt-get install python-nose
sudo easy_install ghp-import
sudo apt-get install pyflakes
sudo apt-get install pychecker
sudo easy_install https://github.com/ponty/paved/zipball/master
sudo apt-get install scrot
sudo apt-get install xvfb
sudo apt-get install xserver-xephyr
sudo apt-get install python-imaging
sudo apt-get install python-sphinx
sudo easy_install sphinxcontrib-programscreenshot
sudo easy_install sphinxcontrib-programoutput
sudo easy_install sphinxcontrib-paverutils
```

6.3 Tasks

Paver is used for task management, settings are saved in pavement.py. Sphinx is used to generate documentation.

print paver settings:

```
paver printoptions
```

clean generated files:

```
paver clean
```

generate documentation under *docs/_build/html*:

```
paver cog pdf html
```

upload documentation to github:

```
paver ghpages
```

run unit tests:

```
paver nose
#or
nosetests --verbose
```

check python code:

```
paver pyflakes
paver pychecker
```

generate python distribution:

```
paver sdist
```

upload python distribution to PyPI:

```
paver upload
```

CHAPTER

SEVEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

p

INDEX

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W